The VIA Isaiah Architecture

G. Glenn Henry, President, Centaur Technology, Inc.

VIA Technologies Inc. started shipping its current family of x86 processors in early 2000. Seven major versions—culminating in the VIA C7 processor—have shipped through the end of 2007. While compatible with the x86 instruction-set architecture, the internal architecture of these processors is very different from other x86 processor designs. This unique internal architecture yields processors that are significantly smaller (lower cost) and use significantly less power than other x86 processors from AMD™ and Intel®.

While the current VIA C7 processors are a perfect fit for many users and applications, a new architecture was needed to keep pace with the rapid introduction of new functions and improved performance from Intel. Accordingly, over the last four years, VIA made a major investment in its U.S.-based processor design subsidiary—Centaur Technology Inc.—to develop a completely new x86 processor architecture. The result is a new architecture, codenamed the *VIA Isaiah Architecture,* that complements current VIA products by offering significantly more function and performance within the same low-power envelope.

The initial products implementing the new VIA Isaiah Architecture will start shipping in the spring of 2008. These processors are built in 65nm technology and provide two to four times the performance of current VIA processors (at the same GHz) without a commensurate increase in cost and with little—if any—increase in power consumption.

This paper summarizes the new VIA Isaiah Architecture with emphasis on the major underlying concepts and some of the unique features.

PERVASIVE DESIGN CONCEPTS

At a high level, all modern superscalar and out-of-order architectures (such as the new VIA Isaiah Architecture) are similar. The details among these architectures differ substantially due to different product objectives, different technologies, and different design philosophies. These detailed differences lead directly to the resulting products having different levels of performance and power consumption. Following are the major design concepts that strongly influenced *our* architecture (and thus, product details):

- **Optimize for mainstream users and applications with high efficiency (low cost and low power consumption) products.** We have always designed our processors to satisfy the needs of *mainstream* users and applications while also providing great efficiency (performance per dollar, and performance per Watt). Using a car analogy, we design an affordable family sedan that gets very good gas mileage. We leave the pickup truck and sports car market to others. In car design, increasing the maximum speed beyond some point, say 100 MPH, is *very* inefficient in terms of cost and mileage. The same basic principle holds true in processor design: obtaining the last "10%" of performance is very costly.

  Thus, our key design principle has always been to eschew being the fastest in the world and instead to focus on being the most efficient—especially considering power consumption—while having sufficient performance for mainstream users and applications. This key principle—obviously different from that of Intel or AMD—was *the* guiding principle driving

most of the detailed design choices and tradeoffs within the VIA Isaiah Architecture. Many detailed architecture choices were driven by our goal of providing much greater performance without any increase in power consumption.

- **Provide a next-generation instruction set**. Year after year the x86 instruction set continues to expand through the addition of new instructions and functions. With time, these extensions become used by software and enable new applications. Accordingly, a key design focus of the VIA CN architecture was to include the latest additions to the x86 instruction set: the 64-bit instruction architecture, the new virtual machine architecture, new SSE instructions, new power management functions, and many other advanced features. Thus, VIA Isaiah Architecture processors can—from day one—support the most recent advances in software.
- **Utilize the newest advances in processor architecture.** Unlike the Intel Core™2 processor designs, which have continually evolved over the last 15 years starting with the Pentium Pro design, the VIA Isaiah Architecture was designed *from scratch* over the last four years. Accordingly, it is optimized for latest advances in processor architecture and is specifically designed to support modern application requirements.

  For example, the architecture was built around the 64-bit instruction set and the new virtual machine functions as opposed to these features being an add-on to an older, lower-function, architecture. As a result, the performance of a CN processor in 64-bit mode has no restrictions or limiting special cases as it does in some other architectures.

  However, as a new architecture, there are many improvements still possible beyond the first implementation. New versions are currently being developed that further improve performance and function through the normal process of "tuning" a new architecture.
- **Facilitate adding new features.** The x86 processor marketplace continually improves performance and adds new instruction features. To support these capabilities, VIA has always focused on the ability to rapidly bring new designs to the marketplace. The VIA Isaiah Architecture has many specific internal design features to facilitate the rapid support of new capabilities.
- **Scalability:** Processors using the VIA Isaiah Architecture are specifically designed to be bus and socket compatible with current VIA processors and able to use the current VIA chipsets. This provides system developers with the ability to span a wide range of performance with one board or system design. This contrasts with the Intel and AMD approach of forcing arbitrary bus and functional incompatibilities between high-end and low-end processors.

ARCHITECTURE HIGHLIGHTS

The VIA Isaiah Architecture implements some of the most modern processor design features in the industry: Following is a summary of the highlights.[1] The next section describes more details.

- **Superscalar and speculative out-of-order:** The *initial* VIA Isaiah Architecture processors can decode three *full* x86 instructions per clock, generate three *fused* micro-ops per clock, issue—speculatively and out-of-order—*seven* execution micro-ops per clock to seven execution ports, and retire three fused micro-ops per clock. [2]

[1] Intel gives some basic architecture features fancy marketing names like "Intel Smart Memory Access". We currently have no fancy name for our features, but often we have the equivalent (or better) features. To help understand our architecture, I'll highlight the equivalent Intel names for some of our features.

[2] Intel calls basic superscalar and speculative out-of-order execution "Wide Dynamic Execution". I can't bring myself to use fancy names for basic and common features like this.

- **Macro-fusion and micro-fusion.** To increase performance and reduce power, specific combinations of two x86 instructions are combined (*fused*) into one executable micro-op (*macro-fusion*). For example, an x86 compare instruction followed by a jump instruction is fused into a single micro-op and executed in one clock. Similarly, multiple micro-ops controlling different execution ports can be fused into a single fused micro-op (*micro fusion*).
- **Sophisticated branch prediction:** The VIA Isaiah Architecture implements a very powerful and unique branch prediction algorithm using *eight* different predictors in two different pipeline stages. The first fetch pipeline stage contains three predictors of conditional branch behavior (each more accurate for a particular type of branch), a predictor of which of these predictors to use, and a separate return predictor. The translate stage (where more information about the instructions is known) contains a return predictor, a conditional branch "overflow" predictor, and a default predictor.

  Each predictor is designed to be more accurate for a particular type of branch at a particular point in the pipeline. The predictors interact and "vote" to obtain the final prediction.
- **Smart cache subsystem:** [3] The VIA Isaiah Architecture includes many innovations in the cache subsystem that lead to very efficient use of *total* cache area (and thus cache power). For example, our 64-KB L1 caches are twice as large as Intel's 32-KB caches and have twice the associativity (16-way versus 8-way). Similarly, our L2 cache (1 MB in the initial product) is 16-way associative while the equivalent-size Intel L2 caches are only eight-way associative. In addition, our L2 cache is "exclusive" versus Intel's "inclusive" design. This means that our L1 caches contents do not reside in our L2 cache, thus increasing the effective size of our L2 cache over the Intel approach.

  Another feature unique to the VIA Isaiah Architecture is that many of the data-prefetch algorithms load prefetched data into a special 64-line prefetch cache as opposed to loading it directly into the L2-cache (such as Intel does). This approach improves the efficiency of the L2 cache and the smaller size is adequate since the useful lifetime of prefetched data is short.
- **Powerful data-prefetch units.** VIA Isaiah Architecture processors implement multiple different data-prefetch mechanisms that analyze data-access patterns and try to load the predicted data from the bus before the reference actually occurs. These prefetch mechanisms run asynchronously from micro-op execution and improve performance significantly for some applications. One mechanism predicts future data use based on past load or store requests that miss in the L1.[4] The prefetch data in this case is loaded into our prefetch cache. Another mechanism is a "streaming prefetcher" that loads prefetched data directly into the L1 cache, just as Intel does.
- **Sophisticated memory access features.** The VIA Isaiah Architecture has many powerful features for improving performance of loads and stores. For example, the "memory disambiguation" algorithm detects loads that hit store data that has not yet stored.[5] This algorithm uses both static address formats as well as history about previous instruction execution. The store-to-load forwarding mechanisms are also very powerful, including, for example, the ability to merge a smaller store into larger load data. The VIA Isaiah Architecture also performs speculative TLB tablewalks and provides very fast handling of unaligned data across a page boundary, and so forth.

[3] While our cache approach is technically different from the "Intel Advanced Smart Cache", we also claim "Advanced Smart Cache" for our innovative approach.

[4] Intel calls this approach "data prefetch logic (DPL)"

[5] Intel calls their version of prefetching and memory disambiguation (next point) "Intel Smart Memory Access".

- **Powerful execution units:** The VIA Isaiah Architecture has *seven* execution ports and can issue up to seven execution micro-ops per clock: two integer, a load, a store address, a store data, a "media", and a multiply micro-op. The media micro-op can be a floating-point add, divide or square root, or a SIMD integer instruction. This unit also overlaps execution of divide and square root with other subsequent media operations; thus, media micro-ops may continue to issue while a divide or square root is executing.

- **High-performance media computation:** The VIA Isaiah Architecture places significant emphasis on high-performance floating-point execution. It can execute four floating-point adds *and* four floating-point multiplies every clock. It uses a completely new algorithm for floating-point adds that results in the lowest floating-point add latency of any x86 processor—*two* clocks for *any* format (SP, DP, DE, packed or scalar). This contrasts with Intel's three or four clocks (depends on the format). Similarly, the floating-point multiplier has the lowest latency of any x86 processor—three clocks for SP multiply, and four for DP and DE. This contrasts with Intel's four and five clocks, respectively.

  In addition, the integer data path for SIMD integer (SSEx) instructions is 128-bits wide, and almost all SSEx instructions—including all shuffles— execute in only one clock.[6]

- **Advanced power and thermal management:** In addition to our usual aggressive dynamic management of active power,[7] VIA Isaiah Architecture processors utilize new low-power circuit techniques. The latest x86 instruction-level power controls are included along with a new "C6" power state where power is turned off to the caches.

  Unique to the VIA Isaiah Architecture are several new VIA Adaptive PowerSaver™ features. They include fine-grained algorithms for adaptively transitioning between performance and voltage states ("P" states) *while the processor continues to run* (Intel stops the bus and execution during these transitions). Yet another feature provides automatic overclocking if the die temperature is low. Another feature allows the processor to automatically maintain the die temperature at a user-specified temperature. Several other new power and thermal management features are provided.

- **Industry-leading security features:** The VIA Isaiah Architecture continues VIA's industry leadership by providing unique x86 instructions for data security. Just like earlier VIA processors, VIA Isaiah Architecture processors contain very high-performance hardware for AES encryption, SHA-1 and SHA-256 secure hashing, and random number generation. In addition, the VIA Isaiah Architecture contains some new (and unique to VIA) features for a very specialized "secure execution mode". These include a secure on-chip memory area, encrypted instruction fetching, and more.

[6] Intel calls having a 128-bit wide datapath "Intel Advanced Digital Media Boost", and their name for a one-clock shuffle capability is "Super Shuffle Engine".

[7] Intel calls the basic features of dynamic power management (such as fine-grained power-gating)—which we have always done— "Intel Intelligent Power capability".

ARCHITECTURE OVERVIEW

Figure 1 shows a conceptual picture of the VIA Isaiah Architecture components and pipeline structure (the actual number of pipeline stages is not shown). There are five major conceptual divisions of the pipeline:

- **Speculative in-order fetching and translation.** These pipeline components fetch x86 instruction bytes and translate them into internal machine instructions, called *micro-ops*. x86 instructions and micro-ops proceed in program order down this portion of the pipeline ("in-order"). The speculative label refers to the fact that the processor may not be actually fetching the correct program instructions (in cases of a branch misprediction, for example).
- **Out-of-order issue and execution.** These pipeline components take the translated micro-ops and issue them to the appropriate execution units. The issue and execution is not necessarily in program order; instructions are issued and executed whenever their inputs are available (thus "out-of-order"). The key to being able to do this out-of-order issue and execution is that information about all micro-ops—and their operands and results—is maintained in a Reorder Buffer (ROB) and a Memory Reorder Buffer (MOB). The result of a micro-op execution is returned to the ROB awaiting its in-order retirement..
- **Program-order retire.** Micro-ops in the ROB that have completed execution are retired in program order. Retirement means that the x86 registers are updated, memory changes are committed, x86 exceptions are taken, and so forth.
- **Memory subsystem.** While the types of memory subsystem components—level-1 caches, a level-2 cache, a bus controller, etc.—are the same as in other architectures, the details of the VIA Isaiah Architecture cache subsystem are quite different.
- **Power & thermal management.** Power management components are distributed among all components of the architecture. Some components (such as clock gating, bus management, and circuit techniques) reduce the operating power consumed by the processor logic. Other mechanisms, unique to VIA Isaiah Architecture, manage dynamic changes to the operating environment: die temperature, applied voltage, and the internal clock multiplier ratio.

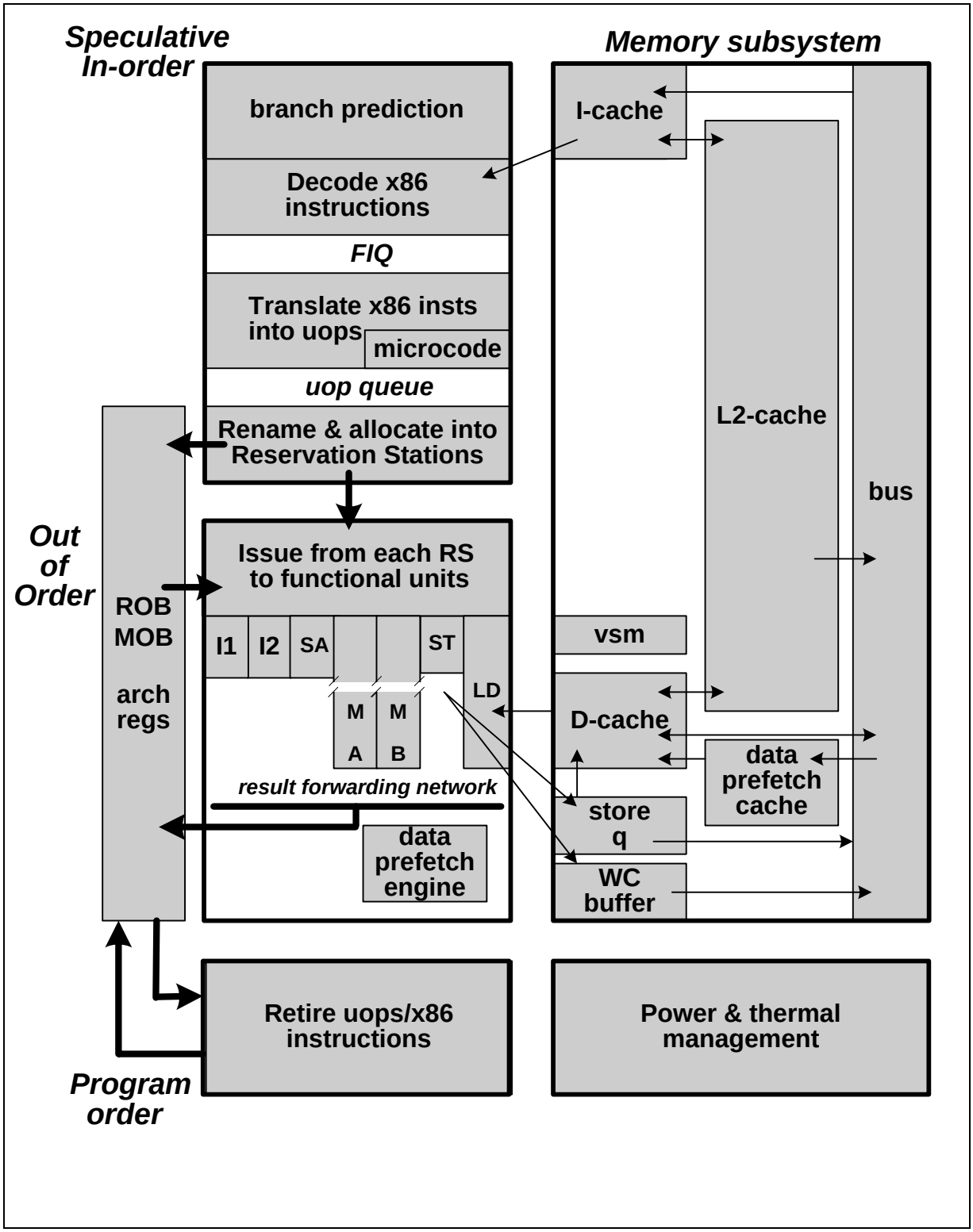
The following more detailed description of the various components refer to the *first implementation* of the VIA Isaiah Architecture.

INSTRUCTION FETCH AND MICRO-OP GENERATION

I-CACHE

Instruction bytes are fetched from the either the bus or from the instruction cache (I-cache). The large I-cache contains 64K bytes and is organized as 16-way set associative. This is twice as large, with twice the associativity, of an Intel Core 2 I-cache.

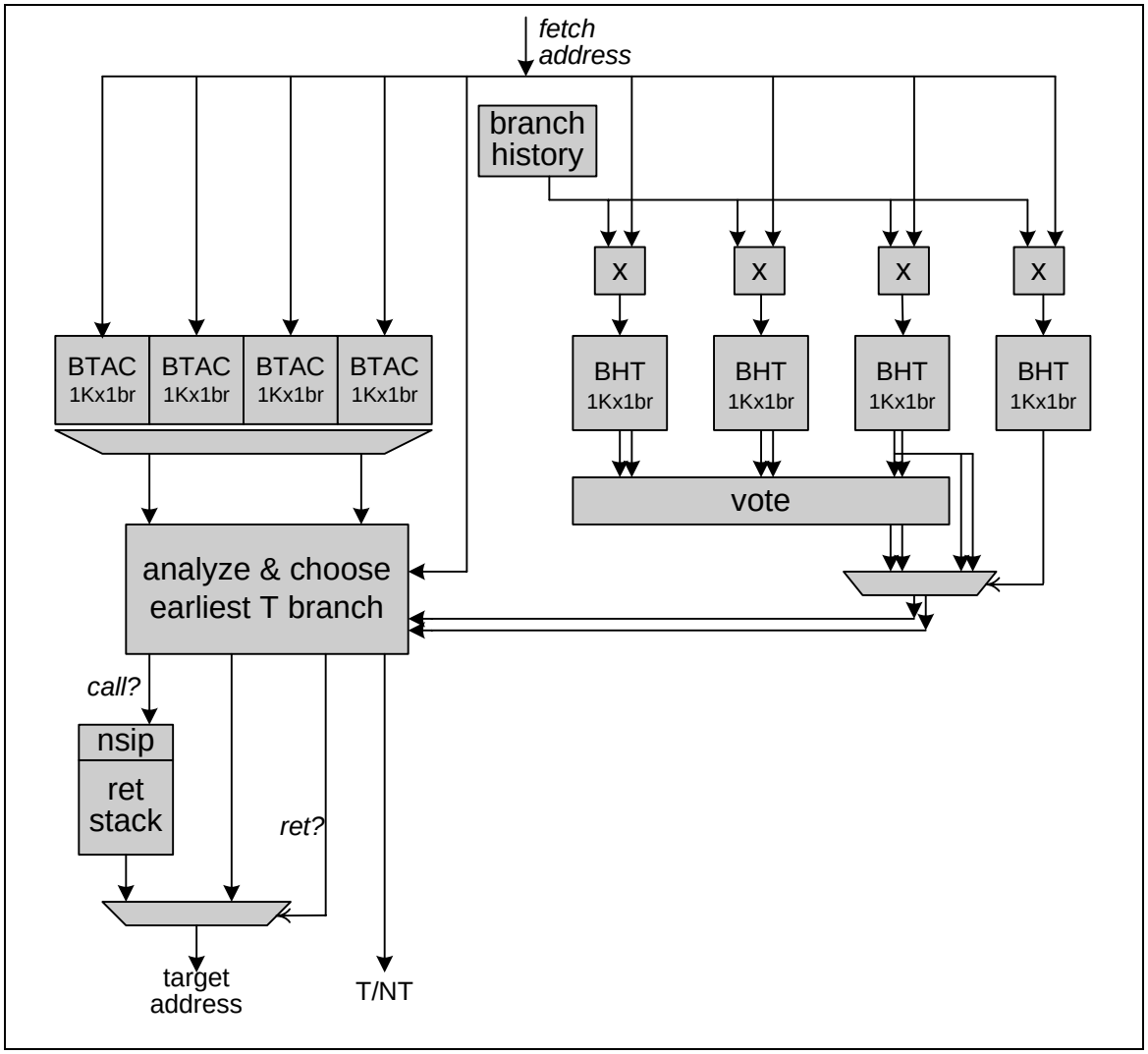
Figure 1. VIA Isaiah Architecture Overview



BRANCH PREDICTION

The primary branch prediction mechanism works in parallel with fetching data from the I-cache. In the initial implementation, two branches can be predicted in each 16-byte "line" fetched from the I-cache. While all modern processors have a branch prediction mechanism at this point in the pipeline (including the previous VIA processors), the details of the VIA Isaiah Architecture branch prediction design are new and unique to the VIA Isaiah Architecture. Figure 2 shows a conceptual picture of the branch prediction unit.

Figure 2. VIA Isaiah Architecture branch Prediction



The Branch Target Address Cache (BTAC) is the table that predicts where branches can occur. Its size is 4K entries organized as four-way associative. Each entry predicts the location and type of a branch. Initially, up to two branches for each 16-byte line are predicted. A Branch History Table (BHT) is a table that predicts the direction of suspected conditional branches. There are four different BHTs. Each contains prediction information about 4Kx2 branches. Each prediction is indexed by a combination of branch history and the instruction address. The details of the index mechanism are different for each of the BHTs since each mechanism is tuned to handle different patterns of likely occurring branches. Three of the BHTs each predict a direction for the branch. The fourth table basically predicts which of these three predictions to use for a given branch based

on the past history for a particular branch. The BHT mechanism predicts direction for two branches in a 16-byte line. A separate return address stack also predicts the address for x86 RETURN instructions.

The x86 instruction translate stage also contains more branch prediction mechanisms (beyond what is shown in Figure 2). A small (2K branches) prediction table handles cases where the branch isn't seen by the BTAC (in case, for example, there are three branches in a 16-byte line). There is also a separate RETURN predictor in this stage along with a default predictor for conditional branches seen for the first time.

In the x86 translate stage, the results of the two stages of branch prediction are compared, and a decision is made about which to use. In the rare case where the translate predictor is chosen over the I-cache predictor, the I-cache prediction is overridden with a branch from the translate stage.

X86 INSTRUCTION TRANSLATION

x86 instructions are translated into executable micro-ops in two stages. The first stage translates x86 instructions into a special internal form. This stage can translate three x86 instructions of *any* length in a single clock, assuming that all instructions *start* within the same 16-byte address range. There is no restriction on the type of x86 instructions or their format—all three x86 instructions can be "complex". Each instruction can have any number of instruction prefixes (including redundant prefixes) as long as the "start within 16 byte" restriction is met. The translated x86 instructions are placed in a large Formatted Instruction Queue (FIQ) queue (16 x86 instructions in the first implementation).

The next stage translates the x86 instructions from the FIQ into micro-ops. Each x86 instruction can be directly translated into zero, one, two or three micro-ops (in the initial implementation) and an optional ROM address. Note that, since x86 instructions require—on average—more than one micro-op, the FIQ queue allows the fetch unit to get ahead of execution. This causes almost all of the branch prediction bubbles to be "absorbed" by this queue and not cause a translation bubble.

Where possible, the translator fuses two separate x86 instructions into one micro-op. For example, the common sequence of a compare instruction followed by a conditional branch instruction is fused into one micro-op. Also, each micro-op generated by the translator can represent functions to be performed by more than one execution unit. This combining of execution unit micro-ops into more powerful micro-ops is called micro-op fusion.

MICROCODE SUBSYSTEM

Complex x86 instructions that require more than three micro-ops use microcode for the additional instructions. The microcode subsystem holds 24K microinstructions plus a powerful "patch" capability allowing updates to microcode in the field. Each ROM microinstruction is translated into up to three separate fused micro-ops (the same micro-ops used by the translator). In addition, the microcode sequencer can perform many types of branches, including conditional branches and call/returns without sending a micro-op to the execution units.

MICRO-OP RENAME

The translator and microcode subsystem can each produce three fused micro-ops (corresponding to up to three x86 instructions) each clock. The translator micro-ops are placed into a micro-op queue. Each clock, the "next" three micro-ops (in program order) enter the rename stage. If the micro-ops taken into rename come from the ROM, the translator continues to translate x86 instructions, depositing the resulting micro-ops into the micro-op queue. This allows the fetch, branch prediction, and translation portion of the pipeline to get far ahead of execution when executing an x86 instruction requiring microcode.

The three incoming fused micro-ops are placed in the reorder buffer (ROB) and are then expanded into up to six executable micro-ops, each of which is targeted for a single execution unit. The registers used in the executable micro-ops are mapped (renamed) into the larger internal set of registers and then placed in the seven micro-op "reservation stations" (RS) associated with the seven issue ports. Up to three micro-ops can be placed into each reservation station each clock. The reservation stations are of different sizes and can hold 76 total micro-ops. The rename stage also contains the memory disambiguation mechanism that attempts to detect store-to-load dependencies to ensure that the store is issued before the dependent load.

MICRO-OP EXECUTION & RETIREMENT

ISSUE

Each clock, the issue logic chooses one micro-op from each of the seven reservation stations to be issued to the corresponding execution port. The choice is made based on whether all the inputs to the micro-op are available, as opposed to the original program order of micro-ops—thus "out-of-order" execution. On every processor clock, instructions are issued such that the pipeline flow of instructions into the execution units is continuous—the issue stage adds no bubbles into the pipeline.

Each execution port feeds a specific set of execution logic. The ports and their associated functions are:

ALU1 PORT

Executes all non-SIMD integer instructions (other than multiply and divide): add/subtracts, logicals, shifts, moves, weird things like bit scans, etc. Also executes microcode branches. All instructions take only one clock to execute.

ALU2 PORT

Same as the ALU1 port except some low-usage functions are missing to allow the ALU2 port to perform x86 branches instead.

STORE ADDRESS PORT

Performs store address computations—and the LEA instruction—and sets up internal store-to-load forwarding tables, etc. Execution of a store address takes one clock.

STORE DATA PORT

Handles the data portion of a store. Normal store data goes into a store queue with 16 entries of 16 bytes each for later delivery to the cache subsystem or bus. There is also a six-line (64-byte lines) write-combining buffer. The store data unit is fully pipelined, and a total of 16 stores may be outstanding at any time. From a program view, the store data operation takes only one clock.

MEDIA-A PORT

Several different execution units are attached to this 128-bit wide port:

- **Floating point add**. Four single-precision SSE floating-point adds, two SSE double-precision floating-point adds, or one x87 add can be executed in parallel by one micro-op. The throughput for all adds is one clock (the unit is fully pipelined) and the latency for all adds is a world-record two clocks.[8]

[8] The Intel Core 2 latency for floating-point adds is three clocks.

- **Simple floating-point operations**. All simple floating-point operations (such as moves or floating-point logical operations) are executed in one clock.
- **SIMD-integer operations.** Almost all SSE integer operations are executed in one clock. The execution data path is 128 bits wide.
- **Divide and square root operations.** A separate multi-clock unit executes both integer and floating-point divides and floating-point square roots. Execution of these "long" operations is completely overlapped (other than the initial clock) with continuing execution of other Media1 port micro-ops such as floating-point adds.
- **Data security operations.** The hardware units that perform the AES encryption and the secure hash algorithms are attached to this port.

MEDIA-B PORT

This port is dedicated to high-performance multiplies, both floating-point and integer. Four single-precision SSE floating-point multiplies, two SSE double-precision floating-point multiplies, or one x87 floating-point multiply can be executed in parallel by one micro-op. Integer and SSE single-precision multiplies are fully pipelined with a world-record latency of three clocks. Double-precision and x87 multiplies have a throughput of two clocks and a latency of four clocks.[9]

The multiply unit also has a fused floating-point multiply-add function that is used by the transcendental algorithms.

LOAD PORT

This port performs loads. Load data can come from the bus (memory), the L1 D-cache, the L2-cache, the data prefetch cache, the store queue, and so forth. The unit is fully pipelined and a total of 16 loads may be outstanding at any time. The latency of the load data in the initial implementation is four clocks. This includes aligning and zero extending the data as required. This unit and the store units are tightly interwoven to provide the aggressive store-to-load forwarding capability of the VIA Isaiah Architecture.

DATA FORWARDING

Each clock, the results from each port are sent to the ROB for use by other micro-ops. Many direct data-forwarding paths also exist between the execution units. The details of which port results get directly forwarded to which ports is highly tuned to reflect the tradeoff between performance and power consumption.

MICRO-OP RETIREMENT

Each clock, the retirement logic looks at the next oldest micro-ops in the ROB and retires those that have completed execution. Up to three fused micro-ops can be retired each clock corresponding to up to three x86 instructions.

CACHE SUBSYSTEM

The cache subsystem comprises the level-1 instruction cache (I-cache), the level-1 data cache (D-cache), the unified level-2 cache (L2-cache), and some specialized caches or buffers. Each of the I- and D-caches contains 64K bytes and is 16-way set associative. This is twice the size and associativity of Intel's Core2 architecture. The L2-cache is organized as 16-way associative and is

[9] The Intel Core 2 latency for floating-point multiplies is four and five clocks for single precision and double precision.

designed to support a wide variety of sizes with minimal implementation effort. The size of the L2-cache in the first VIA Isaiah Architecture implementation is 1 MB, but subsequent processors may have different sizes.

The VIA Isaiah Architecture L2-cache is an "exclusive" cache, as opposed to Intel's "inclusive" L2 cache. This means that data in the level 1 caches are *not* present in the VIA Isaiah Architecture L2-cache. In an inclusive cache, for contrast, all of the data in the L1-caches is also duplicated in the L2-cache. In our design, the only way lines get into the L2-cache is by being evicted from the L1-caches. Thus, the "future value" of the L2 data is high—all data in the L2-cache has been actually used at some time.

This L2-cache difference also causes a difference in how prefetched data is handled. Both the Intel Core 2 and VIA Isaiah Architecture processors have sophisticated data prefetch algorithms that attempt to predict what memory data will be used ahead of when the data is needed. Both architectures can prefetch highly likely predicted data directly into the D-cache. For less likely data, however, the VIA Isaiah Architecture prefetches the data into a dedicated prefetch cache area, while the Intel Core2 prefetches the data into the L2-cache.

The reason for our approach is

- The lifetime of prefetched data is typically short, so a small buffer is sufficient
- Having a separate buffer increases the effective bandwidth of the single ported L2-cache
- It facilitates the asynchronous nature of the prefetcher (it doesn't have to arbitrate for cache access, for example).

The cache subsystem also contains the six-line (a line is 64 bytes here) write-combining buffer to handle write-combining and non-temporal x86 stores. Data is collected and merged together within this buffer and written to main memory via the bus.

A "volatile secure memory" (VSM) area exists that can be loaded or stored with special x86 instructions (available only in a new "secure execution" mode). Data in this area has its own address space and never appears on the bus on in the caches. This is a unique VIA Isaiah Architecture addition to the x86 architecture, intended for use by a specialized "secure code" monitor.

The cache subsystem contains extensive data forwarding mechanisms that forward store data from the queues to subsequent loads with fine granularity, including the ability to handle data-size mismatches between the load and store.

POWER MANAGEMENT LOGIC

In addition to the many other aggressive methods to reduce active power, the VIA Isaiah Architecture implements several unique P-state management features. (A *P-state* is a particular combination of processor voltage and bus-clock multiplier). During execution in modern x86 processors, the processor dynamically changes P-states in order to reduce power consumption for the level of performance needed (as directed by the operating system). VIA has developed a number of new P-state optimizations that our marketing team calls Adaptive PowerSaver Technology™. For example, in Intel processors, the P-state mechanism shifts between the current and a higher target states by:

1. Stopping the bus and processor execution,
2. Changing the voltage to the final target (this can take hundreds of microseconds), and
3. Enabling the bus and execution at the new clock multiplier.

In the VIA Isaiah Architecture, this sequence is done as:

1. Instantaneously change to the "other" PLL (we have two independent PLLs) which is already idling at the next incremental multiplier ratio. The first PLL begins transitioning to the next highest multiplier.
2. Change the voltage one minimum step,
3. Repeat this step-wise iteration until the end point is achieved.

Note that the bus and program execution *remain running during this entire transition*. This is a significant improvement in responsiveness during the many P-state changes mobile processor performs. Another unique VIA Isaiah Architecture mechanism automatically adjusts the P-state voltage based on the die temperature. Assume, for example, that the Isaiah normally requires 1.1 V to run at 2 GHz at its maximum rated die temperature. If, however, during a transition to 2 GHz speed, the die temperature is 20° lower than the maximum rated, then the *Adaptive P-State Control* feature automatically calculates that only, say, 1.0 V is needed to achieve 2 GHz and accordingly only shifts to 1.0 V. Similarly, if the processor is running at its rated 2.0 GHz but the die temperature is lower than the rated maximum, then the *Adaptive Overclocking* feature will automatically raise the voltage and run the part at, say, 2.2 GHz.

Additionally, the VIA Isaiah Architecture implements a *Adaptive Thermal Limit* mechanism. Software system can say, for example, that it wants the die maintained no hotter than 80° (that may be all the heat the system can remove). Using this unique mechanism, the processor will automatically and dynamically adjust P-states such that the target die temperature is not exceeded.

These are only some of the many unique mechanisms provided by the VIA Isaiah Architecture that ensure the best performance per Watt of any x86 processor.

THE FUTURE

What's described in this paper is the first silicon version of the new VIA Isaiah Architecture. Having now seen silicon, however, the designers are infinitely wiser. Thus, new implementations are currently underway that will "tune" the architecture and utilize new technologies to provide future performance increases—but still remaining within the same low power envelope.

AUTHOR BIOGRAPHY

G. Glenn Henry started his career in the computer industry as a computer programmer in 1963 and joined IBM in 1967. During his 21-year career at IBM he was the chief architect and primary development manager of many innovative products such as the first IBM RISC workstation (RT/PC), the first IBM UNIX product (AIX), and the forerunner of the AS/400 (System/38). He was appointed an IBM Fellow in 1985. In 1988, he left IBM to become Dell Computer's first VP of R&D. Working directly for Michael Dell, he subsequently held the positions of Senior VP, Product Group, and Chief Technology Officer. In 1994 he left Dell to work as a consultant to MIPS Technologies to develop a strategy for integrating x86 and MIPS architectures. In 1995 Mr. Henry obtained funding from IDT Inc. and founded Centaur Technology Inc., currently a wholly owned subsidiary of VIA Technologies. Centaur designs low cost, low power x86 processors marketed under the VIA brand name.